High-Level Why-Not Explanations Using Ontologies

High-Level Why-Not Explanations using Ontologies

Balder ten Cate
LogicBlox and UCSC
balder.tencate@gmail.com

Cristina Civili
Sapienza Univ. of Rome
civili@dis.uniroma1.it

Evgeny Sherkhonov
Univ. of Amsterdam
e.sherkhonov@uva.nl

Wang-Chiew Tan
UCSC
tan@cs.ucsc.edu

ABSTRACT

We propose a novel foundational framework for *why-not explanations*, that is, explanations for why a tuple is missing from a query result. Our why-not explanations leverage concepts from an ontology to provide high-level and meaningful reasons for why a tuple is missing from the result of a query.

A key algorithmic problem in our framework is that of *computing a most-general explanation* for a why-not question, relative to an ontology, which can either be provided by the user, or it may be automatically derived from the data and/or schema. We study the complexity of this problem and associated problems, and present concrete algorithms for computing why-not explanations. In the case where an external ontology is provided, we first show that the problem of deciding the existence of an explanation to a why-not question is NP-complete in general. However, the problem is solvable in polynomial time for queries of bounded arity, provided that the ontology is specified in a suitable language, such as a member of the DL-Lite family of description logics, which allows for efficient concept subsumption checking. Furthermore, we show that a most-general explanation can be computed in polynomial time in this case. In addition, we propose a method for deriving a suitable (virtual) ontology from a database and/or a schema, and we present an algorithm for computing a most-general explanation to a why-not question, relative to such ontologies. This algorithm runs in polynomial-time in the case when concepts are defined in a selection-free language, or if the underlying schema is fixed. Finally, we also study the problem of computing *short* most-general explanations, and we briefly discuss alternative definitions of what it means to be an explanation, and to be most general.

Categories and Subject Descriptors

H.2 [**Database Management**]

General Terms

Theory, Algorithms

Keywords

Databases; Why-Not Explanations; Provenance; Ontologies

PODS'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

ACM 978-1-4503-2757-2/15/05 ...$15.00.
http://dx.doi.org/10.1145/2745754.2745765.

1. INTRODUCTION AND RESULTS

An increasing number of databases are derived, extracted, or curated from disparate data sources. Consequently, it becomes more and more important to provide data consumers with mechanisms that will allow them to gain an understanding of data that they are confronted with. An essential functionality towards this goal is the capability to provide meaningful explanations about why data is present or missing form the result of a query. Explanations help data consumers gauge how much trust one can place on the result. Perhaps more importantly, they provide useful information for debugging the query or data that led to incorrect results.

This is particularly the case in scenarios where complex data analysis tasks are specified through large collections of nested views (i.e., views that may be defined in terms of other views). For example, schemas with nested view definitions and integrity constraints capture the core of LogiQL [18, 21, 19] (where view definitions may, in general, involve not only relational operations, but also aggregation, machine learning and mathematical optimization tasks). LogiQL is a language developed and used at LogicBlox [2] for developing data-intensive "self service" applications involving complex data analytics workflows. Similar recent industrial systems include Datomic[1] and Google's Yedalog [16]. In each of these systems, nested view definitions (or, Datalog programs) are used to specify complex workflows to drive data-analytics tasks. Explanations for unexpected query results (such as an unexpected tuple or a missing tuple) are very useful in such settings, since the source of an error can be particularly hard to track.

There has been considerable research on the topic of deriving explanations for why a tuple belongs to the output of a query. Early systems were developed in [3, 28] to provide explanations for answers to logic programs in the context of a deductive database. The presence of a tuple in the output is explained by enumerating all possible derivations, that is, instantiations of the logic rules that derive the answer tuple. In [28], the system also explains missing answers, by providing a partially instantiated rule, based on the missing tuple, and leaving the user to figure out how the rest of the rule would have to be instantiated. In the last decade or so, there has been significant efforts to characterize different notions of provenance (or lineage) of query answers (see, e.g., [15, 20]) which can also be applied to understand why an answer is in the query result.

There have also been extensive studies on the *why-not problem* (e.g., more recent studies include [6, 14, 23, 22, 25, 29]). The why-not problem is the problem of explaining why an answer is missing from the output. Since [28], the *why-not problem* was also studied in [22, 23] in the context of debugging results of data extracted via select-project-join queries, and, subsequently, a larger class of queries that also includes union and aggregation opera-

[1] `www.datomic.com`

tors. Unlike [28] which is geared towards providing explanations for answers and missing answers, the goal in [23] is to propose modifications to underlying database I, yielding another database I' based on the provenance of the missing tuple, constraints, and trust specification at hand, so that the missing tuple appears in the result of the same query q over the updated database I'. In contrast to the *data-centric* approach of updating the database to derive the missing answer, another line of research [8, 14, 29] follows a *query-centric* approach whereby the query q at hand is modified to q' (without modifying the underlying database) so that the missing answer appears in the output of $q'(I)$.

A new take on why-not questions: In this paper, we develop a novel foundational framework for why-not explanations that is principally different from prior approaches. Our approach is neither data-centric nor query-centric. Instead, we derive high-level explanations via an ontology that is either provided, or is derived from the data or schema. Our immediate goal is not to compute repairs of the underlying database or query so that the missing answer would appear in the result. Rather, as in [28], our primary goal is to provide understandable explanations for why an answer is missing from the query result. As we will illustrate, explanations that are based on an ontology have the potential to be high-level and provide meaningful insight to why a tuple is missing from the result. This is because an ontology abstracts a domain in terms of concepts and relationships amongst concepts. Hence, explanations that are based on concepts and relationships from an ontology will embody such high-level abstractions. As we shall describe, our work considers two cases. The first is when an ontology is provided externally, in which case explanations will embody external knowledge about the domain. The second is when an ontology is not provided. For the latter, we allow an ontology to be derived from the schema, and hence explanations will embody knowledge about the domain through concepts and relationships that are defined over the schema.

Formally, an explanation for why a tuple $\overline{a}$ is not among the results of a query $q(I)$, in our framework, is a tuple of concepts from the ontology whose extension includes the missing tuple $\overline{a}$ and, at the same time, does not include any tuples from $q(I)$. For example, a query may ask for all products that each store has in stock, in the form of (product ID, store ID) pairs, from the database of a large retail company. A user may then ask why is the pair *(P0034, S012)* not among the result of the query. Suppose *P0034* refers to a bluetooth headset product and *S012* refers to a particular store in San Francisco. If *P0034* is an instance of a concept *bluetooth headsets* and *S012* is an instance of a concept *stores in San Francisco*, and suppose that no pair (x, y), where x is an instance of *bluetooth headset* and y is an instance of *stores in San Francisco*, belongs to the query result. Then the pair of concepts (*bluetooth headset*, *stores in San Francisco*) is an explanation for the given why-not question. Intuitively, it signifies the fact that "*none of the stores in San Francisco has any bluetooth headsets on stock*".

There may be multiple explanations for a given why-not question. In the above example, this would be the case if, for instance, *S012* belongs also to a more general concept *stores in California*, and that none of the stores in California have bluetooth headsets on stock. Our goal is to compute a *most-general explanation*, that is, an explanation that is not strictly subsumed by any other explanation. We study the complexity of computing a most-general explanation to a why-not question. Formally, we define a *why-not instance* (or, *why-not question*) to be a quintuple $(\mathbf{S}, I, q, Ans, \overline{a})$ where $\mathbf{S}$ is a *schema*, which may include integrity constraints; I is an instance of $\mathbf{S}$; q is a query over $\mathbf{S}$; *Ans* = $q(I)$; and $\overline{a} \notin q(I)$.

As mentioned earlier, a particular scenario where why-not questions easily arise is when querying schemas that include a large collection of views, and where each view may be nested, that is, defined in terms of other views. Our framework captures this setting, since view definitions can be expressed by means of constraints.

Our framework supports a very general notion of an ontology, which we call $\mathbf{S}$-*ontologies*. For a given relational schema $\mathbf{S}$, an $\mathbf{S}$-ontology is a triple $(\mathcal{C}, \sqsubseteq, ext)$ which defines the set of concepts, the subsumption relationship between concepts, and respectively, the extension of each concept w.r.t. an instance of the schema $\mathbf{S}$. We use this general notion of an $\mathbf{S}$-ontology to formalize the key notions of *explanations* and *most-general explanations*, and we show that $\mathbf{S}$-ontologies capture two different types of ontologies.

The first type of ontologies we consider are those that are defined externally, provided that there is a way to associate the concepts in the externally defined ontology to the instance at hand. For example, the ontology may be represented in the form of a Ontology-Based Data Access (OBDA) specification [26]. More precisely, an OBDA specification consists of a set of concepts and subsumption relation specified by means of a description logic terminology, and a set of *mapping assertions* that relates the concepts to a relational database schema at hand. Every OBDA specification induces a corresponding $\mathbf{S}$-ontology. If the concepts and subsumption relation are defined by a TBox in a tractable description logic such as *DL-Lite*$_{\mathcal{R}}$, and the mapping assertions are Global-As-View (GAV) assertions, the induced $\mathbf{S}$-ontology can in fact be computed from the OBDA specification in polynomial time. We then present an algorithm for computing all most-general explanations to a why-not question, given an external $\mathbf{S}$-ontology. The algorithm runs in polynomial time when the arity of the query is bounded, and it executes in exponential time in general. We show that the exponential running time is unavoidable, unless P=NP, because the problem of deciding whether or not there exists an explanation to a why-not question given an external $\mathbf{S}$-ontology is NP-complete in general.

The second type of ontologies that we consider are ontologies that are derived either (a) from a schema $\mathbf{S}$, or (b) from an instance of the schema. In both cases, the concepts of the ontology are defined through concept expressions in a suitable language $L_{\mathbf{S}}$ that we develop. Specifically, our concepts are obtained from the relations in the schema, through selections, projections, and intersection. The difference between the two cases lies in the way the subsumption relation $\sqsubseteq$ is defined. In the former, a concept C is considered to be subsumed in another concept C' if the extension of C is contained in the extension of C' over all instances of the schema. For the latter, subsumption is considered to hold if the extension of C is contained in the extension of C' with respect to the given instance of the schema. The $\mathbf{S}$-ontology induced by a schema $\mathbf{S}$, or instance I, denoted $\mathcal{O}_{\mathbf{S}}$ or $\mathcal{O}_I$, respectively, is typically infinite, and is not intended to be materialized. Instead, we present an algorithm for directly computing a most-general explanation with respect to $\mathcal{O}_I$. The algorithm runs in exponential time in general. However, if the schema is of bounded arity, the algorithm runs in polynomial time. As for computing most-general explanations with respect to $\mathcal{O}_{\mathbf{S}}$, we identify restrictions on the integrity constraints under which the problem is decidable, and we present complexity upper bounds for these cases.

More related work: The use of ontologies to facilitate access to databases is not new. A prominent example is OBDA, where queries are either posed directly against an ontology, or an ontology is used to enrich a data schema against which queries are posed with additional relations (namely, the concepts from the ontology) [9, 26]. Answers are computed based on an open-world assumption and using the mapping assertions and ontology provided by

the OBDA specification. As we described above, we make use of OBDA specifications as a means to specify an external ontology and with a database instance through mapping assertions. However, unlike in OBDA, we consider queries posed against a database instance under the traditional closed-world semantics, and the ontology is used only to derive why-not explanations.

The problems of providing why explanations and why-not explanations have also been investigated in the context of OBDA in [11] and [13], respectively. The why-not explanations of [13] follow the *data-centric* approach to why-not provenance as we discussed earlier where their goal is to modify the assertions that describe the extensions of concepts in the ontology so that the missing tuple will appear in the query result.

There has also been prior work on extracting ontologies from data. For example, in [24], the authors considered heuristics to automatically generate an ontology from a relational database by defining project-join queries over the data. Other examples on ontology extraction from data include publishing relational data as RDF graphs or statements (e.g., D2RQ [10], Triplify [5]). We emphasize that our goal is not to extract and materialize ontologies, but rather, to use an ontology that is derived from data to compute why-not explanations.

Outline: After the preliminaries, in Section 3 we present our framework for why-not explanations. In Section 4 we discuss in detail the two ways of obtaining an $\mathbf{S}$-ontology. In Section 5 we present our main algorithmic results. Finally, in Section 6, we study variatations of our framework, including the problem of producing *short* most-general explanations, and alternative notions of *explanation*, and of what it means to be *most general*.

2. PRELIMINARIES

A *schema* is a pair $(\mathbf{S}, \Sigma)$, where $\mathbf{S}$ is a set $\{R_1, \ldots, R_n\}$ of relation names, where each relation name has an associated arity, and Σ is a set of first-order sentences over $\mathbf{S}$, which we will refer as *integrity constraints*. Abusing the notation, we will write $\mathbf{S}$ for the schema $(\mathbf{S}, \Sigma)$. A *fact* is an expression of the form $R(b_1, \ldots, b_k)$ where $R \in \mathbf{S}$ is a relation of arity k, and for $1 \leq i \leq k$, we have $b_i \in \mathbf{Const}$, where $\mathbf{Const}$ is a countably infinite set of constants. We assume a dense linear order $<$ on $\mathbf{Const}$. An *attribute* A of an k-ary relation name $R \in \mathbf{S}$ is a number i such that $1 \leq i \leq k$. For a fact $R(\overline{b})$ where $\overline{b} = b_1, \ldots, b_k$, we sometimes write $\pi_{A_1,\ldots,A_k}(\overline{b})$ to mean the tuple consisting of the A_1th, ..., A_kth constants in the tuple $\overline{b}$, that is, the value $(b_{A_1}, \ldots, b_{A_k})$. An *atom over* $\mathbf{S}$ is an expression $R(x_1, \ldots, x_n)$, where $R \in \mathbf{S}$ and every $x_i, i \in \{1, \ldots, n\}$ is a variable or a constant.

A *database instance*, or simply an *instance*, I over $\mathbf{S}$ is a set of facts over $\mathbf{S}$ satisfying the integrity constraints Σ. Equivalently, an instance I is a map that assigns to each k-ary relation name $R \in \mathbf{S}$ a finite set of k-tuples over $\mathbf{Const}$ such that the integrity constraints are satisfied. By R^I we denote the set of these tuples. We write $\mathrm{Inst}(\mathbf{S})$ to denote the set of all database instances over $\mathbf{S}$, and $\mathrm{adom}(I)$ to denote the active domain of I, i.e., the set of all constants occurring in facts of I.

Queries A *conjunctive query* (CQ) over $\mathbf{S}$ is a query of the form $\exists \overline{y}.\varphi(\overline{x}, \overline{y})$ where φ is a conjunction of atoms over $\mathbf{S}$. Given an instance I and a CQ q, we write $q(I)$ to denote the set of answers of q over I. In this paper we allow conjunctive queries containing comparisons to constants, that is, comparisons of the form $x \,\mathtt{op}\, c$, where $\mathtt{op} \in \{=, <, >, \leq, \geq\}$ and $c \in \mathbf{Const}$. We show that all upper bounds hold for the case of CQs with such comparisons, and all lower bounds hold without the use of comparisons (unless explicitly specified otherwise). We do *not* allow comparisons between variables.

Integrity constraints In this paper we consider different classes of integrity constraints, including functional dependencies and inclusion dependencies. We also consider UCQ-view definitions and nested UCQ-view definitions, which can be expressed using integrity constraints as well.

A *functional dependency* (FD) on a relation $R \in \mathbf{S}$ is an expression of the form $R : X \rightarrow Y$ where X and Y are subsets of the set of attributes of R. We say that an instance I over $\mathbf{S}$ satisfies the FD if for every $\overline{a}_1$ and $\overline{a}_2$ from R^I if $\pi_A(\overline{a}_1) = \pi_A(\overline{a}_2)$ for every $A \in X$, then $\pi_B(\overline{a}_1) = \pi_B(\overline{a}_2)$ for every $B \in Y$.

An *inclusion dependency* (ID) is an expression of the form

$$R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n]$$

where $R, S \in \mathbf{S}$, each A_i and B_j is an attribute of R and S respectively. We say that an instance I over $\mathbf{S}$ satisfies the ID if

$$\{\pi_{A_1,\ldots,A_n}(\overline{a}) \mid \overline{a} \in R^I\} \subseteq \{\pi_{B_1,\ldots,B_n}(\overline{b}) \mid \overline{b} \in S^I\}.$$

Note that functional and integrity constraints can equivalently be written as first-order sentences [1].

View Definitions To simplify presentation, we treat view defintions as a special case of integrity constraints.

A set of integrity constraints Σ over $\mathbf{S}$ is said to be a *collection of UCQ-view definitions* if there exists a partition $\mathbf{S} = \mathbf{D} \cup \mathbf{V}$ such that for every $P \in \mathbf{V}$, Σ contains exactly one first-order sentence of the form:

$$P(\bar{x}) \leftrightarrow \vee_{i=1}^{k} \varphi_i(\bar{x}), \qquad (*)$$

where each φ_i is a conjunctive query (with comparisons to constants) over $\mathbf{D}$.

Similarly, a set of integrity constraints Σ over $\mathbf{S}$ is said to be a *collection of nested UCQ-view definitions* if there exists a partition $\mathbf{S} = \mathbf{D} \cup \mathbf{V}$ such that for every $P \in \mathbf{V}$, Σ contains exactly one first-order sentence of the form (*), where each φ_i is now allowed to be a conjunctive query over $\mathbf{D} \cup \mathbf{V}$, but subject to the following acyclicity condition. Let us say that $P \in \mathbf{V}$ *depends on* $R \in \mathbf{V}$, if R occurs in the view definition of P, that is, in the sentence of Σ that is of the form (*) for P. We require that the "depends on" relation is acyclic. If, in the view definition of every $P \in \mathbf{V}$, each disjunct φ_i contains at most one atom over $\mathbf{V}$, then we say that Σ is a collection of *linearly* nested UCQ-view definitions.

Note that a collection of nested UCQ-view definitions (in the absence of comparisons) can be equivalently viewed as a non-recursive Datalog program and vice versa [7]. In particular, a collection of linearly nested UCQ-view definitions corresponds to a linear non-recursive Datalog program.

Example 2.1. As an example of a schema, consider $\mathbf{S} = \mathbf{D} \cup \mathbf{V}$ with the integrity constraints in Figure 1. An instance I of the schema $\mathbf{S}$ is given in Figure 2. □

3. WHY-NOT EXPLANATIONS

Next, we introduce our ontology-based framework for explaining why a tuple is not in the output of a query. Our framework is based on a general notion of an ontology. As we shall describe in Section 4, the ontology that is used may be an external ontology (for example, an existing ontology specified in a description logic), or it may be an ontology that is derived from a schema. Both are a special case of our general definition of an $\mathbf{S}$-ontology.

Definition 3.1 ($\mathbf{S}$-ontology)**.** *An* $\mathbf{S}$-ontology *over a relational schema* $\mathbf{S}$ *is a triple* $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$*, where*

Data schema $\mathbf{D}$:

{Cities(name, population, country, continent),
Train-Connections(city_from, city_to)}

View schema $\mathbf{V}$:

{BigCity(name), EuropeanCountry(name),
Reachable(city_from, city_to)}

UCQ-view definitions:

BigCity(x)	$\leftrightarrow$	Cities(x,y,z,w) $\wedge$ $y \geq 5000000$
EuropeanCountry(z)	$\leftrightarrow$	Cities(x,y,z,w) $\wedge$ w = Europe
Reachable(x,y)	$\leftrightarrow$	Train-Connections(x,y) $\vee$ (Train-Connections(x,z) $\wedge$ Train-Connections (z,y))

Functional and inclusion dependencies:

country	$\rightarrow$	continent
BigCity[name]	$\subseteq$	Train-Connections[city_from]
Train-Connections[city_from]	$\subseteq$	Cities[name]
Train-Connections[city_to]	$\subseteq$	Cities[name]

Figure 1: Example of a schema S.

Cities

name	population	country	continent
Amsterdam	779,808	Netherlands	Europe
Berlin	3,502,000	Germany	Europe
Rome	2,753,000	Italy	Europe
New York	8,337,000	USA	N.America
San Francisco	837,442	USA	N.America
Santa Cruz	59,946	USA	N.America
Tokyo	13,185, 000	Japan	Asia
Kyoto	1,400,000	Japan	Asia

Train-Connections

city_from	city_to
Amsterdam	Berlin
Berlin	Rome
Berlin	Amsterdam
New York	San Francisco
San Francisco	Santa Cruz
Tokyo	Kyoto

BigCity

name
New York
Tokyo

EuropeanCountry

name
Netherlands
Germany
Italy

Reachable

city_from	city_to
Amsterdam	Berlin
Berlin	Rome
Berlin	Amsterdam
New York	San Francisco
San Francisco	Santa Cruz
Tokyo	Kyoto
Amsterdam	Rome
Amsterdam	Amsterdam
Berlin	Berlin
New York	Santa Cruz

Figure 2: Example of an instance I of S.

- *$\mathcal{C}$ is a possibly infinite set, whose elements are called* concepts,
- *$\sqsubseteq$ is a pre-order (i.e., a reflexive and transitive binary relation) on $\mathcal{C}$, called the* subsumption relation, *and*
- *$ext : \mathcal{C} \times \mathrm{Inst}(\mathbf{S}) \rightarrow \wp(\mathbf{Const})$ is a polynomial-time computable function that will be used to identify instances of a concept in a given database instance ($\wp(\mathbf{Const})$ denotes the powerset of $\mathbf{Const}$).*

More precisely, we assume that ext is specified by a Turing machine that, given $C \in \mathcal{C}$, $I \in \mathrm{Inst}(\mathbf{S})$ and $c \in \mathbf{Const}$, decides in polynomial time if $c \in ext(C, I)$.

A database instance $I \in \mathrm{Inst}(\mathbf{S})$ is consistent *with $\mathcal{O}$ if, for all $C_1, C_2 \in \mathcal{C}$ with $C_1 \sqsubseteq C_2$, we have $ext(C_1, I) \subseteq ext(C_2, I)$.*

An example of an $\mathbf{S}$-ontology $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ is shown in Figure 3, where the concept subsumption relation $\sqsubseteq$ is depicted by means of a Hasse diagram. Note that, in this example, $ext(C, I)$ is independent of the database instance I (and, as a consequence, every $\mathbf{S}$-instance is consistent with $\mathcal{O}$). In general, this is not the case (for example, the extension of a concept may be determined through mapping assertions, cf. Section 4.1).

We define our notion of an ontology-based explanation next.

Definition 3.2 (Explanation)**.** *Let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be an $\mathbf{S}$-ontology, I an $\mathbf{S}$-instance consistent with $\mathcal{O}$. Let q be an m-ary query over $\mathbf{S}$, and $\overline{a} = (a_1, \ldots, a_m)$ a tuple of constants such that $\overline{a} \notin q(I)$. Then a tuple of concepts $(C_1, \ldots, C_m)$ from $\mathcal{C}^m$ is called an* explanation *for $\overline{a} \notin q(I)$ with respect to $\mathcal{O}$ (or an* explanation *in short) if:*

- *for every $1 \leq i \leq m$, $a_i \in ext(C_i, I)$, and*
- *$(ext(C_1, I) \times \ldots \times ext(C_m, I)) \cap q(I) = \emptyset$.*

In other words, an explanation is a tuple of concepts whose extension includes the missing tuple $\overline{a}$ (and thus explains $\overline{a}$) but, at the same time, it does not include any tuple in $q(I)$ (and thus does not explain any tuple in $q(I)$). Intuitively, the tuple of concepts is an explanation that is orthogonal to existing tuples in $q(I)$ but relevant for the missing tuple $\overline{a}$, and thus forms an explanation for why $\overline{a}$ is not in $q(I)$. There can be multiple explanations in general and the "best" explanations are the ones that are the most general.

Definition 3.3 (Most-general explanation)**.** *Let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be an $\mathbf{S}$-ontology, and let $E = (C_1, \ldots, C_m)$ and $E' = (C'_1, \ldots, C'_m)$ be two tuples of concepts from $\mathcal{C}^m$.*

- *We say that E is* less general *than E' with respect to $\mathcal{O}$, denoted as $E \leq_{\mathcal{O}} E'$, if $C_i \sqsubseteq C'_i$ for every $i, 1 \leq i \leq m$.*
- *We say that E is* strictly less general *than E' with respect to $\mathcal{O}$, denoted as $E <_{\mathcal{O}} E'$, if $E \leq_{\mathcal{O}} E'$, and $E' \not\leq_{\mathcal{O}} E$.*
- *We say that E is a* most-general explanation *for $\overline{a} \notin q(I)$ if E is an explanation for $\overline{a} \notin q(I)$, and there is no explanation E' for $\overline{a} \notin q(I)$ such that $E' >_{\mathcal{O}} E$.*

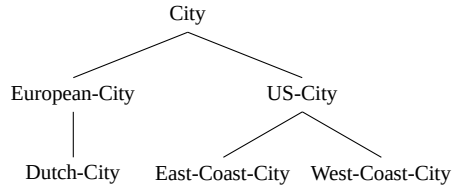
As we will formally define in Section 5, a *why-not problem* asks the question: "why is the tuple $(a_1, \ldots, a_m)$ not in the output of a query q over an instance I of schema $\mathbf{S}$?" The following example illustrates the notions of explanations and most-general explanations in the context of a why-not problem.

Example 3.4. Consider the instance $I_{\mathbf{D}}$ of the relational schema $\mathbf{S}$ = {Cities(name, population, country, continent), Train-Connections(city_from, city_to)} shown in Figure 2.

Suppose q is the query $\exists z$. Train-Connections(x, z) $\wedge$ Train-Connections(z, y). That is, the query asks for all pairs of cities that are connected via a city. Then $q(I)$ returns tuples $\{\langle$Amsterdam, Rome$\rangle, \langle$Amsterdam, Amsterdam$\rangle, \langle$Berlin, Berlin$\rangle, \langle$New York, Santa Cruz$\rangle\}$. A user may ask why is the tuple $\langle$Amsterdam, New York$\rangle$ not in the result of $q(I)$ (i.e., why is $\langle$Amsterdam, New York$\rangle \notin q(I)$?). Based on the $\mathbf{S}$-ontology defined in Figure 3, we can derive the following explanations for $\langle$Amsterdam, New York$\rangle \notin q(I)$:

$E_1 = \langle$Dutch-City, East-Coast-City$\rangle$
$E_2 = \langle$Dutch-City, US-City$\rangle$
$E_3 = \langle$European-City, East-Coast-City$\rangle$
$E_4 = \langle$European-City, US-City$\rangle$

E_1 is the simplest explanation, i.e., the one we can build by looking at the lower level of the hierarchy in our $\mathbf{S}$-ontology. Each subsequent explanation is more general than at least one of the prior explanations w.r.t. to the $\mathbf{S}$-ontology. In particular, we have $E_4 >_{\mathcal{O}} E_2 >_{\mathcal{O}} E_1$, and $E_4 >_{\mathcal{O}} E_3 >_{\mathcal{O}} E_1$. Thus, the most-general explanation for why $\langle$Amsterdam, New York$\rangle \notin q(I)$ with respect to our $\mathbf{S}$-ontology is E_4, which intuitively informs that the reason is because Amsterdam is a city in Europe while New York is a city in the US (and hence, they are not connected by train). Note that all the other possible combinations of concepts are not explanations because they intersect with $q(I)$. □



$ext(\text{City}, I)$ = {Amsterdam, Berlin, Rome, New York, San Francisco, Santa Cruz, Tokyo, Kyoto}
$ext(\text{European-City}, I)$ = {Amsterdam, Berlin, Rome}
$ext(\text{Dutch-City}, I)$ = {Amsterdam}
$ext(\text{US-City}, I)$ = {New York, San Francisco, Santa Cruz}
$ext(\text{East-Coast-City}, I)$ = {New York}
$ext(\text{West-Coast-City}, I)$ = {Santa Cruz, San Francisco}

Figure 3: Example ontology.

As we will see in Example 4.9, there may be more than one most-general explanations in general.

Generalizing the above example, we can informally define the problem of explaining why-not questions via ontologies as follows: *given an instance I of schema $\mathbf{S}$, a query q over $\mathbf{S}$, an $\mathbf{S}$-ontology $\mathcal{O}$ (consistent with I) and a tuple $\overline{a} \notin q(I)$, compute a most-general explanation for $\overline{a} \notin q(I)$, if one exists, w.r.t. $\mathcal{O}$.* As we shall describe in Section 5, in addition to the above problem of computing one most-general explanation, we will also investigate the corresponding decision problem that asks whether or not an explanation for a why-not problem exists, and whether or not a given tuple of concepts is a most-general explanation for a why-not problem. In our framework, the $\mathbf{S}$-ontology $\mathcal{O}$ may be given explicitly as part of the input, or it may be derived from a given database instance or a given schema. We will introduce the different scenarios by which an ontology may be obtained in the next section, before we describe our algorithms for computing most-general explanations in Section 5.

4. OBTAINING ONTOLOGIES

In this section we discuss two approaches by which $\mathbf{S}$-ontologies may be obtained. The first approach allows one to leverage an external ontology, provided that there is a way to relate a concept in the ontology to a database instance. In this case, the set $\mathcal{C}$ of concepts is specified through a description logic such as $\mathcal{ALC}$ or *DL-Lite*; $\sqsubseteq$ is a partial order on the concepts defined in the ontology, and the function *ext* may be given through *mapping assertions*. The second approach considers an $\mathbf{S}$-ontology that is derived from a specific database instance, or from a schema. This approach is useful as it allows one to define an ontology to be used for explaining why-not questions in the absence of an external ontology.

In either case, we study the complexity of deriving such $\mathbf{S}$-ontologies based on the language on which concepts are defined, the subsumption between concepts, and the function ext, which is defined according to the semantics of the concept language.

4.1 Leveraging an external ontology

We first consider the case where we are given an external ontology that models the domain of the database instance, and a relationship between the ontology and the instance. We will illustrate in particular how *description logic ontologies* are captured as a special case of our framework.

In what follows, our exposition borrows notions from the Ontology-Based Data Access (OBDA) framework. Specifically, we will make crucial use of the notion of an *OBDA specification* [17], which consists of a description logic ontology, a relational schema, and a collection of mapping assertions. To keep the exposition simple, we restrict our discussion to one particular description logic, called *DL-Lite*$_\mathcal{R}$, which is a representative member of the *DL-Lite* family of description logics [12]. *DL-Lite*$_\mathcal{R}$ is the basis for the OWL 2 QL[2] profile of OWL 2, which is a standard ontology language for Semantic Web adopted by W3C. As the other languages in the *DL-Lite* family, *DL-Lite*$_\mathcal{R}$ exhibits a good trade off between expressivity and complexity bounds for important reasoning tasks such as subsumption checking, instance checking and query answering.

TBox and Mapping Assertions. In the description logic literature, an ontology is typically formalized as a TBox (Terminology Box), which consists of finitely many *TBox axioms*, where each TBox axiom expresses a relationship between concepts. Alongside TBoxes, ABoxes (Assertion Boxes) are sometimes used to describe the extension of concepts. To simplify the presentation, we do not consider ABoxes here.

Definition 4.1 (*DL-Lite*$_\mathcal{R}$)**.** *Fix a finite set Φ_C of* "atomic concepts" *and a finite set Φ_R of* "atomic roles".

- *The* concept expressions *and* role expressions *of DL-Lite*$_\mathcal{R}$ *are defined as follows:*

Basic concept expression:	$B ::= A \mid \exists R$
Basic role expression:	$R ::= P \mid P^-$
Concept expressions:	$C ::= B \mid \neg B$
Role expressions	$E ::= R \mid \neg R$

  *where $A \in \Phi_C$ and $P \in \Phi_R$. Formally, a (Φ_C, Φ_R)-*interpretation *$\mathcal{I}$ is a map that assigns to every atomic concept in Φ_C a unary relation over $\mathbf{Const}$ and to every atomic role in Φ_R a binary relation over $\mathbf{Const}$. The map $\mathcal{I}$ naturally extends to arbitrary concept expressions and role expressions:*

$$\mathcal{I}(P^-) = \{(x,y) \mid (y,x) \in \mathcal{I}(P)\} \quad \mathcal{I}(\exists P) = \pi_1(\mathcal{I}(P))$$
$$\mathcal{I}(\neg P) = \mathbf{Const}^2 \setminus \mathcal{I}(P) \quad \mathcal{I}(\neg A) = \mathbf{Const} \setminus \mathcal{I}(A)$$

  Observe that $\mathcal{I}(\exists P^-) = \pi_2(\mathcal{I}(P))$.

- *A* TBox (Terminology Box) *is a finite set of* TBox axioms *where each TBox axiom is an inclusion assertion of the form $B \sqsubseteq C$ or $R \sqsubseteq E$, where B is a basic concept expression, C is a concept expression, R is a basic role expression and E is a role expression. An (Φ_C, Φ_R)-interpretation $\mathcal{I}$* satisfies *a TBox if for each axiom $X \sqsubseteq Y$, it holds $\mathcal{I}(X) \subseteq \mathcal{I}(Y)$.*

- *For concept expressions C_1, C_2 and a TBox $\mathcal{T}$, we say that C_1* is subsumed by *C_2* relative to *$\mathcal{T}$ (notation: $\mathcal{T} \models C_1 \sqsubseteq C_2$) if, for all interpretations $\mathcal{I}$ satisfying $\mathcal{T}$, we have that $\mathcal{I}(C_1) \subseteq \mathcal{I}(C_2)$.*

An example of a *DL-Lite*$_\mathcal{R}$ TBox is given at the top of Figure 4. For convenience, we have listed next to each TBox axiom, its equivalent semantics in first-order notation.

Next we describe what mapping assertions are. Given an ontology and a relational schema, we can specify mapping assertions to relate the ontology language to the relational schema, which is similar to how mappings are used in OBDA [26]. In general, mapping assertions are first order sentences over the schema $\mathbf{S} \cup \Phi_C \cup \Phi_R$ that express relationships between the symbols in $\mathbf{S}$ and those in Φ_C and Φ_R. Among the different schema mapping languages that can be used, we restrict our attention, for simplicity, to the class of *Global-As-View (GAV) mapping assertions* (*GAV mapping assertions or GAV constraints* or *GAV source-to-target tgds*).

[2] `http://www.w3.org/TR/owl2-profiles/#OWL_2_QL`

DL-Lite TBox axiom	(first-order translation)
EU-City $\sqsubseteq$ City	$\forall x$ EU-City$(x) \rightarrow$ City(x)
Dutch-City $\sqsubseteq$ EU-City	$\forall x$ Dutch-City$(x) \rightarrow$ EU-City(x)
N.A.-City $\sqsubseteq$ City	$\forall x$ N.A.-City$(x) \rightarrow$ City(x)
EU-City $\sqsubseteq \neg$ N.A.-City	$\forall x$ EU-City$(x) \rightarrow \neg$N.A.-City(x)
US-City $\sqsubseteq$ N.A.-City	$\forall x$ US-City$(x) \rightarrow$ N.A.-City(x)
City $\sqsubseteq \exists$ hasCountry	$\forall x$ City$(x) \rightarrow \exists y$ hasCountry(x, y)
Country $\sqsubseteq \exists$ hasContinent	$\forall x$ Country$(x) \rightarrow \exists y$ hasContinent(x, y)
$\exists$hasCountry$^- \sqsubseteq$ Country	$\forall x\ (\exists y$ hasCountry$(y, x)) \rightarrow$ Country(x)
$\exists$hasContinent$^- \sqsubseteq$ Continent	$\forall x\ (\exists y$ hasContinent$(y, x)) \rightarrow$ Continent(x)
$\exists$connected $\sqsubseteq$ City	$\forall x\ (\exists y$ connected$(x, y)) \rightarrow$ City(x)
$\exists$connected$^- \sqsubseteq$ City	$\forall x\ (\exists y$ connected$(y, x)) \rightarrow$ City(x)

GAV mapping assertions (universal quantifiers omitted for readability):

Cities$(x, z, w,$ "Europe"$)$	$\rightarrow$	EU-City(x)
Cities$(x, z,$ "Netherlands"$, w)$	$\rightarrow$	Dutch-City(x)
Cities$(x, z, w,$ "N.America"$)$	$\rightarrow$	N.A.-City(x)
Cities$(x, z,$ "USA"$, w)$	$\rightarrow$	US-City(x)
Cities(x, y, z, w)	$\rightarrow$	Continent(w)
Cities(x, k, y, w)	$\rightarrow$	hasCountry(x,y)
Cities(x, k, w, y)	$\rightarrow$	hasContinent(x,y)
Train-Connection(x, y), Cities(x, x_1, x_2, x_3), Cities(y, y_1, y_2, y_3)	$\rightarrow$	connected(x,y)

Figure 4: Example DL-Lite ontology with mapping assertions.

Definition 4.2 (GAV mapping assertions)**.** *A GAV mapping assertion over $(\mathbf{S}, (\Phi_C \cup \Phi_R))$ is a first-order sentence ψ of the form*

$$\forall \vec{x}\ (\varphi_1(\vec{x_1}), \cdots, \varphi_n(\vec{x_n})) \rightarrow \psi(\vec{x})$$

where $\vec{x} \subseteq \vec{x_1} \cup \ldots \cup \vec{x_n}$, $\varphi_1, \ldots, \varphi_n$ are atoms over $\mathbf{S}$ and ψ is an atomic formula of the form $A(x_i)$ (for $A \in \Phi_C$) or $P(x_i, x_j)$ (for $P \in \Phi_R$). Let I be an $\mathbf{S}$-instance and $\mathcal{I}$ an (Φ_C, Φ_R)-interpretation. We say that the pair $(I, \mathcal{I})$ satisfies *the GAV mapping assertion (notation: $(I, \mathcal{I}) \models \psi$) if it holds that for any tuple of elements $\bar{a}$ from $adom(I)$, with $\bar{a} = \bigcup_{1 \leq k \leq n} \bar{a_k}$, if $I \models \varphi_1(\bar{a_1}), \ldots, \varphi_n(\bar{a_n})$, then $a_i \in \mathcal{I}(A)$, with $a_i \in \bar{a}$ (if $\psi = A(x_i)$) or $(a_i, a_j) \in \mathcal{I}(P)$, with $a_i, a_j \in \bar{a}$ (if $\psi = P(x_i, x_j)$).*

Intuitively, a GAV mapping assertion associates a conjunctive query over $\mathbf{S}$ to an element (concept or atomic role) of the ontology. A set of GAV mapping assertions associates, in general, a union of conjunctive queries to an element of the ontology. Examples of GAV mapping assertions are given at the bottom of Figure 4.

OBDA induced ontologies

Definition 4.3 (OBDA specification)**.** *Let $\mathcal{T}$ be a TBox, $\mathbf{S}$ a relational schema, and $\mathcal{M}$ a set of mapping assertions from $\mathbf{S}$ to the concepts of $\mathcal{T}$. We call the triple $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ an* OBDA specification.

An (Φ_C, Φ_R)-interpretation $\mathcal{I}$ is said to be a solution *for an $\mathbf{S}$-instance I with respect to the OBDA specification $\mathcal{B}$ if the pair $(I, \mathcal{I})$ satisfies all mapping assertions in $\mathcal{M}$ and $\mathcal{I}$ satisfies $\mathcal{T}$.*

Note that our notion of an OBDA specification is a special case of the one given in [17], where we do not consider view inclusion dependencies. Also, as mentioned earlier, our OBDA specifications in this paper assume that $\mathcal{T}$ is a *DL-Lite*$_\mathcal{R}$ TBox and $\mathcal{M}$ is a set of GAV mappings. These restrictions allow us to achieve good complexity bounds for explaining why-not questions with ontologies. In particular, it is not hard to see that, for the OBDA specifications we consider, every $\mathbf{S}$-instance I has a solution.

Theorem 4.1. *([12, 26]) Let $\mathcal{T}$ be a DL-Lite*$_\mathcal{R}$ *TBox.*

1. *There is a* PTIME*-algorithm for deciding subsumption. That is, given $\mathcal{T}$ and two concepts C_1, C_2, decide if $\mathcal{T} \models C_1 \sqsubseteq C_2$.*
2. *There is an algorithm that, given an OBDA specification $\mathcal{B}$, an instance I over $\mathbf{S}$ and a concept C, computes $certain(C, I, \mathcal{B}) = \bigcap\{\mathcal{I}(C) \mid \mathcal{I}$ is a solution for I w.r.t. $\mathcal{B}\}$. For a fixed OBDA specification, the algorithm runs in* PTIME *(*AC^0 *in data complexity).*

Every OBDA specification induces an $\mathbf{S}$-ontology as follows.

Definition 4.4. *Every OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ where $\mathcal{T}$ is a DL-Lite*$_\mathcal{R}$ *TBox and $\mathcal{M}$ is a set of GAV mappings gives rise to an $\mathbf{S}$-ontology where:*

- *$\mathcal{C}_{\mathcal{O}_\mathcal{B}}$ is the set of all basic concept expressions occurring in $\mathcal{T}$;*
- *$\sqsubseteq_{\mathcal{O}_\mathcal{B}} = \{(C_1, C_2) \mid \mathcal{T} \models C_1 \sqsubseteq C_2\}$*
- *$ext_{\mathcal{O}_\mathcal{B}}$ is the polynomial-time computable function given by $ext_{\mathcal{O}_\mathcal{B}}(C, I) = \bigcap\{\mathcal{I}(C) \mid \mathcal{I}$ is a solution for I w.r.t. $\mathcal{B}\}$*

Note that the fact that $ext_{\mathcal{O}_\mathcal{B}}$ is the polynomial-time computable follows from Theorem 4.1.

We remarked earlier that, for the ODBA specifications $\mathcal{B}$ that we consider, it holds that every input instance has a solution. It follows that every input instance I is consistent with the corresponding $\mathbf{S}$-ontology $\mathcal{O}_\mathcal{B}$.

Theorem 4.2. *The $\mathbf{S}$-ontology $\mathcal{O}_\mathcal{B} = (\mathcal{C}_{\mathcal{O}_\mathcal{B}}, \sqsubseteq_{\mathcal{O}_\mathcal{B}}, ext_{\mathcal{O}_\mathcal{B}})$ can be computed from a given OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ in* PTIME *if $\mathcal{T}$ is a DL-Lite*$_\mathcal{R}$ *TBox and $\mathcal{M}$ is a set of GAV mappings.*

We are now ready to illustrate an example where a why-not question is explained via an external ontology.

Example 4.5. Consider the OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$ where $\mathcal{T}$ is the TBox consisting of the *DL-Lite*$_\mathcal{R}$ axioms given in Figure 4, $\mathbf{S}$ is the schema from Example 3.4, and $\mathcal{M}$ is the set of mapping assertions given in Figure 4. These together induce an $\mathbf{S}$-ontology $\mathcal{O}_\mathcal{B} = (\mathcal{C}_{\mathcal{O}_\mathcal{B}}, \sqsubseteq_{\mathcal{O}_\mathcal{B}}, ext_{\mathcal{O}_\mathcal{B}})$. The set $\mathcal{C}_{\mathcal{O}_\mathcal{B}}$ consists of the following basic concept expressions:

City, EU-City, N.A.-City, Dutch-City,
US-City, Country, Continent,
$\exists$ hasCountry, $\exists$ hasCountry$^-$, $\exists$ hasContinent,
$\exists$ hasContinent$^-$, $\exists$ connected, $\exists$ connected$^-$.

The set $\sqsubseteq_{\mathcal{O}_\mathcal{B}}$ includes the pairs of concepts of the TBox $\mathcal{T}$ given in Figure 4. We use the mappings to compute the extension of each concept in $\mathcal{C}_{\mathcal{O}_\mathcal{B}}$ using the instance I on the left of Figure 2. We list a few extensions here:

$ext_{\mathcal{O}_\mathcal{B}}$(City, I)	=	{Amsterdam, Berlin, Rome, New York, San Francisco, Santa Cruz, Tokyo, Kyoto}
$ext_{\mathcal{O}_\mathcal{B}}$(EU-City, I)	=	{Amsterdam, Berlin, Rome}
$ext_{\mathcal{O}_\mathcal{B}}$(N.A.-City, I)	=	{New York, San Francisco, Santa Cruz}
$ext_{\mathcal{O}_\mathcal{B}}$($\exists$hasCountry$^-$, I)	=	{Netherlands, Germany, Italy, USA, Japan}
$ext_{\mathcal{O}_\mathcal{B}}$($\exists$connected, I)	=	{Amsterdam, Berlin, New York}

Now consider the query $q(x, y) = \exists z$. Train-Connections(x, z) $\wedge$ Train-Connections(z, y), and $q(I)$ as in Example 3.4. As before, we would like to explain why is $\langle$Amsterdam, New York$\rangle \notin q(I)$. This time, we use the induced $\mathbf{S}$-ontology $\mathcal{O}_\mathcal{B}$ described above to derive explanations for $\langle$Amsterdam, New York$\rangle \notin q(I)$:

E_1	= $\langle$EU-City, N.A.-City$\rangle$	E_2	= $\langle$Dutch-City,N.A.-City$\rangle$
E_3	= $\langle$EU-City, US-City$\rangle$	E_4	= $\langle$Dutch-City,US-City$\rangle$

Among the four explanations above, E_1 is the most general. □

4.2 Ontologies derived from a schema

We now move to the second approach where an ontology is derived from an instance or a schema. The ability to derive an ontology through an instance or a schema is useful in the context where

an external ontology is unavailable. To this purpose we first introduce a simple but suitable concept language that can be defined over the schema $\mathbf{S}$.

Specifically, our concept language, denoted as $L_{\mathbf{S}}$, makes use of two relational algebra operations, projection (π) and selection (σ). We first introduce and motivate the language. We will then describe our complexity results for testing whether one concept is subsumed by another, and for obtaining an ontology from a given instance or a schema. We will make use of these results later on in Section 5.2 and Section 5.3.

Definition 4.6 (The Concept Language $L_{\mathbf{S}}$)**.** *Let $\mathbf{S}$ be a schema. A concept in $L_{\mathbf{S}}$ is an expression C defined by the following grammar.*

$$D ::= R \mid \sigma_{A_1 \mathtt{op}\, c_1, \ldots, A_n \mathtt{op}\, c_n}(R)$$
$$C := \top \mid \{c\} \mid \pi_A(D) \mid C \sqcap C$$

In the above, R is a predicate name from $\mathbf{S}$, $A, A_1, \ldots, A_n$ are attributes in R, not necessarily distinct, $c, c_1, \ldots, c_n \in \mathbf{Const}$, and each occurrence of $\mathtt{op}$ is a comparison operator belonging to $\{=, <, >, \leq, \geq\}$. For $\mathbf{C} = \{C_1, \ldots, C_k\}$ a finite set of concepts, we denote by $\sqcap\mathbf{C}$ the conjunction $C_1 \sqcap \ldots \sqcap C_k$. If $\mathbf{C}$ is empty, we take $\sqcap\mathbf{C}$ to be $\top$.

Given a finite set of constants $\mathcal{K} \subset \mathbf{Const}$, we define $L_{\mathbf{S}}[\mathcal{K}]$ as the concept language $L_{\mathbf{S}}$ whose concept expressions only use constants from $\mathcal{K}$. By selection-free $L_{\mathbf{S}}$*, we mean the language $L_{\mathbf{S}}$ where σ is not allowed. Similarly, by* intersection-free $L_{\mathbf{S}}$*, we mean the language $L_{\mathbf{S}}$ where $\sqcap$ is not allowed, and by $L_{\mathbf{S}}^{\min}$, we mean the minimal concept language $L_{\mathbf{S}}$ where both σ and $\sqcap$ are not allowed.*

Observe that the $L_{\mathbf{S}}$ grammar defines a concept in the form $C_1 \sqcap \ldots \sqcap C_n$ where each C_i is $\top$ or $\{c\}$ or $\pi_A(R)$ or $\pi_A(\sigma_{A_1 \mathtt{op}\, c_1, \ldots, A_n \mathtt{op}\, c_n}(R))$. A concept of the form $\{c\}$ is called a *nominal*. A nominal $\{c\}$ is the "most specific" concept for the constant c. Given a tuple $\overline{a}$ that is not in the output, the corresponding tuple of nominal concepts forms a default, albeit trivial, explanation for why not $\overline{a}$.

As our next example illustrates, even though our concept language $L_{\mathbf{S}}$ appears simple, it is able to naturally capture many intuitive concepts over the domain of the database.

Example 4.7. We refer back to our schema $\mathbf{S}$ in Figure 1. Suppose we do not have access to an external ontology such as the one given in Example 3.4. We show that even so, we can still construct meaningful concepts directly from the database schema using the concept language described above. We list a few semantic concepts that can be specified with $L_{\mathbf{S}}$ in Figure 5, where we also show the corresponding SELECT-FROM-WHERE style expressions and intuitive meaning. □

Example 4.7 shows that, even though $L_{\mathbf{S}}$ is a simple language where concepts are essentially intersections of unary projections of relations and nominals, it is already sufficiently expressive to capture natural concepts that can be used to build meaningful explanations. It is worth noting that, for minor extensions of the language $L_{\mathbf{S}}$, such as with $\neq$-comparisons and disjunction, the notion of a most-general explanation becomes trivial, in the sense that, for each why-not question, there is a most-general explanation that essentially enumerates all tuples in the query answer.

By using $L_{\mathbf{S}}$, we are able to define an ontology whose atomic concepts are derived from the schema itself. This approach allows us to provide explanations using a vocabulary that is already familiar to the user. We believe that this leads to intuitive and useful why-not explanations.

If we view each expression $\pi_A(D)$ as an atomic concept, then the language $L_{\mathbf{S}}$ corresponds to a very simple concept language, whose concepts are built from atomic concepts and nominals using only intersection. In this sense, $L_{\mathbf{S}}$ can be considered to be a fragment of *DL-Lite*$_{core,\sqcap}$ with nominals (also known as *DL-Lite*$_{horn}$ [4]), i.e., the description logic obtained by enriching *DL-Lite*$_{core}$ (the simplest language in the DL-Lite family) with conjunction.

The precise semantics of $L_{\mathbf{S}}$ is as follows. Given a concept C that is defined in $L_{\mathbf{S}}$ and an instance I over $\mathbf{S}$, the extension of C in I, denoted by $[\![C]\!]^I$, is inductively defined below. Intuitively, the extension of C in I is the result of evaluating the query associated with C over I.

$$\begin{array}{ll}
[\![R]\!]^I & = R^I \\
[\![\sigma_{A_1 \mathtt{op}_1 c_1, \ldots, A_n \mathtt{op}_n c_n}(R)]\!]^I & = \{\bar{b} \in R^I \mid \pi_{A_i}(\bar{b}) \mathtt{op}_i c_i, 1 \leq i \leq n\} \\
[\![\top]\!]^I & = \mathbf{Const} \\
[\![\{c\}]\!]^I & = \{c\} \\
[\![\pi_A(D)]\!]^I & = \pi_A([\![D]\!]^I) \\
[\![C_1 \sqcap C_2]\!]^I & = [\![C_1]\!]^I \cap [\![C_2]\!]^I
\end{array}$$

The notion of when one concept is subsumed by another is defined according to the extensions of the concepts. There are two notions, corresponding to concept subsumption w.r.t. an instance or subsumption w.r.t. a schema. More precisely, given two concepts C_1, C_2,

- we say that C_2 *subsumes* C_1 *w.r.t. an instance* I (notation: $C_1 \sqsubseteq_I C_2$) if $[\![C_1]\!]^I \subseteq [\![C_2]\!]^I$.
- we say that C_2 *subsumes* C_1 *w.r.t. a schema* $\mathbf{S}$ (notation: $C_1 \sqsubseteq_{\mathbf{S}} C_2$), if for every instance I of $\mathbf{S}$, we have that $C_1 \sqsubseteq_I C_2$.

We are now ready to define the two types of ontologies, which are based on the two notions of concept subsumption described above, that can be derived from an instance or a schema.

Definition 4.8 (Ontologies derived from a schema)**.** *Let $\mathbf{S}$ be a schema, and let I be an instance of $\mathbf{S}$. Then the ontologies derived from $\mathbf{S}$ and I are defined respectively as*

- $\mathcal{O}_{\mathbf{S}} = (L_{\mathbf{S}}, \sqsubseteq_{\mathbf{S}}, ext)$ *and*
- $\mathcal{O}_I = (L_{\mathbf{S}}, \sqsubseteq_I, ext)$,

where ext is the function given by $ext(C, I') = [\![C]\!]^{I'}$ for all instances I' over $\mathbf{S}$. By $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ we denote the ontology $(L_{\mathbf{S}}[\mathcal{K}], \sqsubseteq_{\mathbf{S}}, ext)$, and by $\mathcal{O}_I[\mathcal{K}]$ we denote the ontology $(L_{\mathbf{S}}[\mathcal{K}], \sqsubseteq_I, ext)$.

It is easy to verify that the subsumption relations $\sqsubseteq_{\mathbf{S}}$ and $\sqsubseteq_I$ are indeed pre-orders (i.e., reflexive, and transitive relations), and that, for every fixed schemas $\mathbf{S}$, the function $[\![C]\!]^{I'}$ is polynomial-time computable. Hence, the above definition is well-defined even though the ontologies obtained in this way are typically infinite. From the definition, it is easy to verify that if $C_1 \sqsubseteq_{\mathbf{S}} C_2$, then $C_1 \sqsubseteq_I C_2$.

The following result about deciding $\sqsubseteq_I$ is immediate, as one can always execute the queries that are associated with the concepts and then test for subsumption, which can be done in polynomial time.

Proposition 4.1. *The problem of deciding, given an instance I of a schema $\mathbf{S}$ and given two $L_{\mathbf{S}}$ concept expressions C_1, C_2, whether $C_1 \sqsubseteq_I C_2$, is in* PTIME.

On the other hand, the complexity of deciding $\sqsubseteq_{\mathbf{S}}$ depends on the type of integrity constraints that are used in the specification of $\mathbf{S}$. Table 1 provides a summary of relevant complexity results.

Theorem 4.3. *Let $\mathcal{W}$ be one of the different classes of schemas with integrity constraints listed in Table 1. The complexity of the problem to decide, given a schema $\mathbf{S}$ in $\mathcal{W}$ and two $L_{\mathbf{S}}$ concept*

$L_{\mathbf{S}}$ **concept expression**	**SELECT-FROM-WHERE formulation**	**Intuitive meaning**
$\pi_{\text{name}}(\text{Cities})$	name from Cities	City
$\pi_{\text{name}}(\sigma_{\text{continent="Europe"}}(\text{Cities}))$	name from Cities where continent="Europe"	European City
$\pi_{\text{name}}(\sigma_{\text{continent="N.America"}}(\text{Cities}))$	name from Cities where continent="N.America"	N.American City
$\pi_{\text{name}}(\sigma_{\text{population}>1000000}(\text{Cities}))$	name from Cities where population>1000000	Large City
$\pi_1(\text{BigCity})$	name from BigCity	name of BigCity
{"Santa Cruz"}	"Santa Cruz"	Santa Cruz
$\pi_{\text{name}}(\sigma_{\text{population}<1000000}(\text{Cities}))\sqcap$ $\pi_{\text{city\_to}}(\sigma_{\text{city\_from}=\text{Amsterdam}}(\text{Reachable}))$	name from Cities where population<1000000 AND city_from from Reachable where city_to=Amsterdam	Small City that is reachable from Amsterdam.

Figure 5: Example of concepts specified in $L_{\mathbf{S}}$.

Constraints	Complexity of subsumption for $L_{\mathbf{S}}$
UCQ-view def. (no comparisons)	NP-complete
UCQ-view def.	Π_2^P-complete
linearly nested UCQ-view def.	Π_2^P-complete
nested UCQ-view def.	CONEXPTIME-complete
FDs	in PTIME
IDs	? (in PTIME for selection-free $L_{\mathbf{S}}$)
IDs + FDs	Undecidable

All stated lower bounds already hold for $L_{\mathbf{S}}^{\min}$ concept expressions.

Table 1: Complexity of concept subsumption.

expressions C_1, C_2, whether $C_1 \sqsubseteq_{\mathbf{S}} C_2$, is as indicated in the second column of the corresponding row in Table 1.

For example, given two concepts C_1, C_2, and a schema $(\mathbf{S}, \Sigma)$ where Σ is a collection of nested UCQ-view definitions, the complexity of deciding $C_1 \sqsubseteq_{\mathbf{S}} C_2$ is CONEXPTIME-complete. The lower bound already holds for concepts specified in $L_{\mathbf{S}}^{\min}$. We conclude this section with an analysis of the number of distinct concepts that can be formulated in a given concept language and an example that illustrates explanations that can be computed from such derived ontologies.

Proposition 4.2. *Given a schema $\mathbf{S}$ and a finite set of constants $\mathcal{K} \subset \mathbf{Const}$, the number of unique concepts (modulo logical equivalence)*

- *in $L_{\mathbf{S}}^{\min}[\mathcal{K}]$ is polynomial in the size of $\mathbf{S}$ and $\mathcal{K}$,*
- *in selection-free or intersection-free $L_{\mathbf{S}}[\mathcal{K}]$ is single exponential in the size of $\mathbf{S}$ and $\mathcal{K}$.*
- *in $L_{\mathbf{S}}[\mathcal{K}]$ is double exponential in the size of $\mathbf{S}$ and $\mathcal{K}$.*

Example 4.9. Let $\mathbf{S}$ and I be the schema and instance from Figure 1 and Figure 2. Suppose the concept language $L_{\mathbf{S}}$ is used to define among others the concepts from Figure 5. The following concept subsumptions can be derived from $\mathbf{S}$. Note that subsumption $\sqsubseteq_{\mathbf{S}}$ implies $\sqsubseteq_I$.

$$\begin{array}{lcl}
\pi_{\text{name}}(\sigma_{\text{continent="Europe"}}(\text{Cities})) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{name}}(\text{Cities}) \\
\pi_{\text{name}}(\sigma_{\text{population}>7000000}(\text{Cities})) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{name}}(\text{BigCity}) \\
\pi_{\text{name}}(\text{BigCity}) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{name}}(\text{Cities}) \\
\pi_{\text{name}}(\text{BigCity}) & \sqsubseteq_{\mathbf{S}} & \pi_{\text{city_from}}(\text{Train-Connections})
\end{array}$$

The first and second subsumptions follow from definitions. The third one holds because according to Π, a BigCity is a city with population more than 5 million. The fourth subsumption follows from the inclusion dependency that each BigCity must have a train departing from it. There are subsumptions that hold in $\mathcal{O}_I$ but not in $\mathcal{O}_{\mathbf{S}}$. For instance,

$$\pi_{\text{city\_to}}(\sigma_{\text{city\_from}=\text{Amsterdam}}(\text{Reachable})) \sqsubseteq_I \pi_{\text{city\_to}}(\sigma_{\text{city\_from}=\text{Berlin}}(\text{Reachable})),$$

holds w.r.t. $\mathcal{O}_I$, where I is the instance given in Figure 2, but does not hold w.r.t $\mathcal{O}_{\mathbf{S}}$, since one can construct an instance where not all cities that are reachable from Amsterdam are reachable from Berlin.

We now give examples of most-general explanations w.r.t. $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. As before, let $q(x, y) = \exists z.$ Train-Connections$(x, z) \wedge$ Train-Connections(z, y) be a query with $q(I) = \{\langle$Amsterdam, Rome$\rangle, \langle$Amsterdam, Amsterdam$\rangle, \langle$Berlin, Berlin$\rangle, \langle$New York, Santa Cruz $\rangle\}$. We would like to explain why $\langle$Amsterdam, New York$\rangle \notin q(I)$ using the derived ontologies $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. Note that if E is an explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$, then it is also an explanation w.r.t. $\mathcal{O}_I$ and vice versa. Some possible explanations are:

$$\begin{array}{l}
E_1 = \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \\
\qquad \pi_{\text{city_from}}(\sigma_{\text{city_to = San Francisco}}(\text{Train-Connections}))\rangle \\
E_2 = \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \\
\qquad \pi_{\text{name}}(\sigma_{\text{continent=N.America}}(\text{Cities}))\rangle \\
E_3 = \langle \pi_{\text{city_to}}(\sigma_{\text{city_from = Berlin}}(\text{Reachable})), \\
\qquad \pi_{\text{city_from}}(\sigma_{\text{city_to = Santa Cruz}}(\text{Reachable}))\rangle \\
E_4 = \langle \{\text{Amsterdam}\}, \pi_{\text{name}}(\sigma_{\text{population}>7000000}(\text{Cities}))\rangle \\
E_5 = \langle \pi_{\text{name}}(\sigma_{\text{country=Netherlands}}(\text{Cities})), \\
\qquad \pi_{\text{name}}(\text{BigCity}) \sqcap \pi_{\text{name}}(\sigma_{\text{continent=N.America}}(\text{Cities}))\rangle \\
E_6 = \langle \{\text{Amsterdam}\}, \{\text{New York}\}\rangle \\
E_7 = \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \pi_{\text{name}}(\text{BigCity})\}\rangle \\
E_8 = \langle \pi_{\text{name}}(\sigma_{\text{continent=Europe}}(\text{Cities})), \\
\qquad \pi_{\text{name}}(\sigma_{\text{population}>7000000}(\text{Cities}))\}\rangle
\end{array}$$

For example, E_1 states the reason is that Amsterdam is a European city and New York is a city that has a train connection to San Francisco, and there is no train connection between such cities via a city. The trivial explanation E_6 is less general than any other explanation w.r.t $\mathcal{O}_{\mathbf{S}}$ (and $\mathcal{O}_I$ too). It can be verified that E_2 and E_7 are most-general explanations w.r.t both $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. In particular, $E_2 >_{\mathcal{O}_I} E_5$ and $E_2 \geq_{\mathcal{O}_I} E_3$, but $E_2 \not>_{\mathcal{O}_{\mathbf{S}}} E_5$ and $E_2 \not>_{\mathcal{O}_{\mathbf{S}}} E_3$ since there might be an instance of $\mathbf{S}$ where Netherlands is not in Europe or where Berlin is reachable from a non-european city. □

In general, if E is an explanation w.r.t. $\mathcal{O}_I$ then E is also an explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$, and vice versa. The following proposition also describes the relationship between most-general explanations w.r.t $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$.

Proposition 4.3. *Let $\mathbf{S}$ be a schema, and let I be an instance of $\mathbf{S}$.*

(i) Every explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$ is an explanation w.r.t. $\mathcal{O}_I$ and vice versa.

(ii) A most-general explanation w.r.t $\mathcal{O}_{\mathbf{S}}$ is not necessarily a most-general explanation w.r.t. $\mathcal{O}_I$, and likewise vice versa.

Proof. The statement (i) follows from Definition 3.2 and the definition of ext for $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$. That is, ext is the same on the input instance I for both $\mathcal{O}_{\mathbf{S}}$ and $\mathcal{O}_I$, and the conditions of Definition 3.2 use only the value of ext on I. Going back to Example 4.9, E_1 is a most-general explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$, but it is not a most-general explanation w.r.t. $\mathcal{O}_I$ (since E_3 is a strictly more general explanation than E_1 w.r.t. $\mathcal{O}_I$). Thus, the first direction of (ii) holds. For the other direction of (ii), consider E_8 which is a most-general

explanation w.r.t. $\mathcal{O}_I$. But it holds that $E_7 >_{\mathcal{O}_\mathbf{S}} E_8$ and E_7 is an explanation. Note that E_7 and E_8 are equivalent w.r.t. $\mathcal{O}_I$. □

5. ALGORITHMS FOR COMPUTING MOST-GENERAL EXPLANATIONS

Next, we formally introduce the ontology-based why-not problem, which was informally described in Section 3, and we define algorithms for computing most-general explanations. We start by defining the notion of a why-not instance (or why-not question).

Definition 5.1 (Why-not instance)**.** *Let* $\mathbf{S}$ *be a schema,* I *an instance of* $\mathbf{S}$*,* q *an* m*-ary query over* I *and* $\overline{a} = (a_1, \ldots, a_m)$ *a tuple of constants such that* $\overline{a} \notin q(I)$*. We call the quintuple* $(\mathbf{S}, I, q, Ans, \overline{a})$*, where* $Ans = q(I)$*, a* why-not instance *or a* why-not question*.*

In a why-not instance, the answer set *Ans* of q over I is assumed to have been computed already. This corresponds closely to the scenario under which why-not questions are posed where the user requests explanations for why a certain tuple is missing in the output of a query, which is computed a priori. Note that since *Ans*=$q(I)$ is part of a why-not instance, the complexity of evaluating q over I does not affect the complexity analysis of the problems we study in this paper. In addition, observe that although a query q is part of a why-not instance, the query is not directly used in our derivation of explanations for why-not questions with ontologies. However, the general setup accomodates the possibility to consider q directly in the derivation of explanations and this is part of our future work.

We will study the following algorithmic problems concerning most-general explanations for a why-not instance.

Definition 5.2. *The* EXISTENCE-OF-EXPLANATION *problem is the following decision problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$ *consistent with* I*, does there exist an explanation for* $\overline{a} \notin Ans$ *w.r.t.* $\mathcal{O}$*?*

Definition 5.3. *The* CHECK-MGE *problem is the following decision problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$*, an* $\mathbf{S}$*-ontology* $\mathcal{O}$ *consistent with* I*, and a tuple of concepts* $(C_1, \ldots, C_n)$*, is the given tuple of concepts a most-general explanation w.r.t.* $\mathcal{O}$ *for* $\overline{a} \notin Ans$*?*

Definition 5.4. *The* COMPUTE-ONE-MGE *problem is the following computational problem: given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$ *consistent with* I*, find a most-general explanation w.r.t.* $\mathcal{O}$ *for* $\overline{a} \notin Ans$*, if one exists.*

Note that deciding the existence of an explanation w.r.t. a finite $\mathbf{S}$-ontology is equivalent to deciding existence of a most-general explanation w.r.t. the same $\mathbf{S}$-ontology.

Thus, our approach to the why-not problem makes use of $\mathbf{S}$-ontologies. In particular, our notion of a "best explanation" is a *most-general explanation*, which is defined with respect to an $\mathbf{S}$-ontology. We study the problem in three flavors: one in which the $\mathbf{S}$-ontology is obtained from an external source, and thus it is part of the input, and two in which the $\mathbf{S}$-ontology is not part of the input, and is derived, respectively, from the schema $\mathbf{S}$, or from the instance I.

5.1 External Ontology

We start by studying the case of computing ontology-based why-not explanations w.r.t. an external $\mathbf{S}$-ontology. We first study the complexity of deciding whether or not there exists an explanation w.r.t. an external $\mathbf{S}$-ontology.

Theorem 5.1.

1. *The problem* CHECK-MGE *is solvable in* PTIME*.*
2. *The problem* EXISTENCE-OF-EXPLANATION *is* NP*-complete. It remains* NP*-complete even for bounded schema arity.*

Intuitively, to check if a tuple of concepts is a most-general explanation, we can first check in PTIME if it is an explanation. Then, for each concept in the explanation, we can check in PTIME if it is subsumed by some other concept in $\mathcal{O}$ such that by replacing it with this more general concept, the tuple of concepts remains an explanation. The membership in NP is due to the fact that we can guess a tuple of concepts of polynomial size and verify in PTIME that it is an explanation. The lower bound is by a reduction from the SET COVER problem. Our reduction uses a query of unbounded arity and a schema of bounded arity. As we will show in Theorem 5.2, the problem is in PTIME if the arity of the query is fixed.

In light of the above result, we define an algorithm, called the EXHAUSTIVE SEARCH ALGORITHM, which is an EXPTIME algorithm for solving the COMPUTE-ONE-MGE problem.

Algorithm 1: EXHAUSTIVE SEARCH ALGORITHM

Input: a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, where $\overline{a} = (a_1, \ldots, a_m)$, a finite $\mathbf{S}$-ontology $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$

Output: the set of most-general explanations for $\overline{a} \notin Ans$ wrt $\mathcal{O}$

1 Let $\mathcal{C}(a_i) = \{C \in \mathcal{C} \mid a_i \in ext(C, I)\}$ for all $i, 1 \leq i \leq m$
2 Let $\mathcal{X} = \{(C_1, \ldots, C_m) \mid C_i \in \mathcal{C}(a_i)$ and $(ext(C_1, I) \times \ldots \times ext(C_m, I)) \cap Ans = \emptyset\}$
3 **foreach** *pair of explanations* E_1,$E_2 \in \mathcal{X}$, $E_1 \neq E_2$ **do**
4 **if** $E_1 >_\mathcal{O} E_2$ **then**
5 remove E_2 from $\mathcal{X}$
6 return $\mathcal{X}$

This algorithm first generates the set of all possible explanations, and then iteratively reduces the set by removing the tuples of concepts that are less general than some tuple of concepts in the set. In the end, only most-general explanations are returned. At first, in line 1, for each element of the tuple $\overline{a} = (a_1, \ldots, a_m)$, we build the set $\mathcal{C}(a_i)$ containing all the concepts in $\mathcal{C}$ whose extension contains a_i. Then, in line 2, we build the set of all possible explanations by picking a concept in $\mathcal{C}(a_i)$ for each position in $\overline{a}$, and by discarding the ones that have a non empty intersection with the answer set Ans. Finally, in lines 3-5, we remove from the set those explanations that have a strictly more general explanation in the set.

We now show that EXHAUSTIVE SEARCH ALGORITHM is correct (i.e. it outputs the set of all most-general explanations for the given why-not instance w.r.t. to the given $\mathbf{S}$-ontology), and runs in exponential time in the size of the input.

Theorem 5.2. *Let the why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and the* $\mathbf{S}$*-ontology* $\mathcal{O}$ *be an input to* EXHAUSTIVE SEARCH ALGORITHM *and let* $\mathcal{X}$ *be the corresponding output. The following hold:*

1. $\mathcal{X}$ *is the set of all most-general explanations for* $\overline{a} \notin Ans$ *(modulo equivalence);*
2. EXHAUSTIVE SEARCH ALGORITHM *runs in* EXPTIME *in the size of the input (in* PTIME *if we fix the arity of the input query).*

Theorem 5.2, together with Theorem 4.2, yields the following corollary (recall that, by construction of $\mathcal{O}_\mathcal{B}$, it holds that every input instance I is consistent with $\mathcal{O}_\mathcal{B}$).

Corollary 5.5. *There is an algorithm that takes as input a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and an OBDA specification $\mathcal{B} = (\mathcal{T}, \mathbf{S}, \mathcal{M})$, where $\mathcal{T}$ is a DL-Lite$_\mathcal{R}$ TBox and $\mathcal{M}$ is a set of GAV mappings, and computes all the most-general explanations for $\overline{a} \notin Ans$ w.r.t. the $\mathbf{S}$-ontology $\mathcal{O}_\mathcal{B}$ in* ExpTime *in the size of the input (in* PTime *if the arity of the q is fixed)* .

5.2 Ontologies from an instance

We now study the why-not problem w.r.t. an $\mathbf{S}$-ontology O_I that is derived from an instance. First, note that the presence of nominals in the concept language guarantees a trivial answer for the Existence-of-Explanation w.r.t. $\mathcal{O}_I$ problem. An explanation always exists, namely the explanation with nominals corresponding to the constants of the tuple $\overline{a}$. In fact, a *most-general explanation* always exists, as follows from the results below.

Definition 5.6. *The* Compute-one-MGE w.r.t. $\mathcal{O}_I$ *is the following computational problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, find a most-general explanation w.r.t. $\mathcal{O}_I$ for $\overline{a} \notin Ans$, where $\mathcal{O}_I$ is the $\mathbf{S}$-ontology that is derived from I, as defined in Section 4.2.*

First, we state an important proposition, that underlies the correctness of the algorithms that we will present. The following proposition shows that, when we search for explanations w.r.t. $\mathcal{O}_I$, we can always restrict our attention to a particular finite restriction of this ontology.

Proposition 5.1. *Let $(\mathbf{S}, I, q, Ans, \overline{a})$ be a why-not instance. If E is an explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I$ (resp. $\mathcal{O}_\mathbf{S}$), then there exists an explanation E' for $\overline{a} \notin Ans$ such that $E <_{\mathcal{O}_I[\mathcal{K}]} E'$ (resp. $E <_{\mathcal{O}_\mathbf{S}[\mathcal{K}]} E'$), where $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$ and each constant in E' belongs to $\mathcal{K}$.*

In our proof, we iteratively reduce the number of constants occurring in the explanation. That is, for every explanation E with concepts containing constants outside of $\mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$, we produce a new explanation E' which is more general than E and which contains less constants outside of $\mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$.

Notice that since, in principle, it is possible to materialize the ontology $\mathcal{O}_I[\mathcal{K}]$ (i.e., to explicitly compute all the concepts $\mathcal{C}$ in the ontology, the subsumption relation $\sqsubseteq_I$, and the extension ext), the Exhaustive Search Algorithm, together with Proposition 5.1, give us a method for solving Compute-one-MGE w.r.t. $\mathcal{O}_I$. In particular, given a schema, Exhaustive Search Algorithm solves Compute-one-MGE w.r.t. $\mathcal{O}_I$ in 2ExpTime (in ExpTime if the arity of q is fixed). This is because to find a most-general explanation w.r.t $\mathcal{O}_I$, it is sufficient to restrict to the concept language $L_\mathbf{S}[\mathcal{K}]$ and its fragments, where $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$. Then Compute-one-MGE w.r.t. $\mathcal{O}_I$ is solvable in 2ExpTime follows from the fact that the $\mathbf{S}$-ontology $\mathcal{O}_I[\mathcal{K}]$ is computable in at most 2ExpTime.

We now present a more effective algorithm for solving Compute-one-MGE w.r.t. $\mathcal{O}_I$. (See Algorithm 2.) We start by introducing the notion of a *least upper bound* of a set of constants X w.r.t. an instance I, denoted by $\mathsf{lub}_I(X)$. This, intuitively, corresponds to the most-specific concept whose extension contains all constants of X. We first consider the case in which $\mathsf{lub}_I(X)$ is expressed using selection-free $L_\mathbf{S}$ concepts. The following lemma states two important properties of $\mathsf{lub}_I(X)$ that are crucial for the correctness of Algorithm 2.

Lemma 5.1. *Given an instance I of schema $\mathbf{S}$ and a set of constants X, we can compute in polynomial time a selection-free $L_\mathbf{S}$ concept, denoted $\mathsf{lub}_I(X)$, that is the smallest concept whose extension contains all the elements in X definable in the language. In particular, the following hold:*

1. *$X \subseteq ext(\mathsf{lub}_I(X), I)$,*
2. *there is no concept C' in selection-free $L_\mathbf{S}$ such that $C' \sqsubset_I \mathsf{lub}_I(X)$ and $X \subseteq ext(C', I)$.*

We are now ready to introduce the algorithm. We will start with a high-level description of the idea behind it. The algorithm navigates through the search space of possible explanations using an incremental search strategy and makes use of the above defined notion of lub. We start with an explanation that has, in each position, the lub of the constant (i.e., nominal) that occurs in that position. Then, we try to construct a more general explanation by expanding the set of constants considered by each lub.

Notice that Incremental Search Algorithm produces explanations which are tuples of conjunctions of concepts. Therefore it produces an explanation whose concepts are concept expressions in the language $L_\mathbf{S}$ or selection-free $L_\mathbf{S}$. We will study the behavior of the algorithm in each of these cases separately.

Algorithm 2: Incremental Search Algorithm

Input: a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$
Output: a most-general explanation for $\overline{a} \notin Ans$ wrt $\mathcal{O}_I$

```
Let K = adom(I) ∪ {a_1, ..., a_m}
Let X = (X_1, ..., X_m) s.t. each X_j = {a_j}. // support set
Let E = (C_1, ..., C_m) s.t. each C_j = lub_I(X_j). // first candidate explanation
foreach 1 ≤ j ≤ m do
    foreach b ∈ adom(I) \ ext(E_j, I) do
        X'_j = X_j ∪ {b}
        Let C'_j = lub_I(X'_j) // a more general concept in position j
        Let E' := (C_1, ..., C'_j, ... C_m) // a more general explanation
        if E' ∩ Ans = ∅ then
            E := E'
            X := (X_1, ..., X'_j, ... X_m)
return E
```

First, we focus on the case in which Incremental Search Algorithm produces most-general explanations using selection-free $L_\mathbf{S}$ concepts. We show that the algorithm is correct, i.e., that it outputs an explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I$, and that it runs in polynomial time with selection-free $L_\mathbf{S}$.

Theorem 5.3 (Correctness and running time of Incremental Search Algorithm)**.** *Let the why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ be an input to* Incremental Search Algorithm *and E the corresponding output. The following holds:*

1. *E is a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is selection-free $L_\mathbf{S}$;*
2. Incremental Search Algorithm *runs in* PTime *in the size of the input.*

Now we extend our analysis of Incremental Search Algorithm to the general case in which it works with $L_\mathbf{S}$. First, we state an analogue of Lemma 5.1 for $L_\mathbf{S}$.

Lemma 5.2. *Given an instance I of $\mathbf{S}$ and a set of constants X, we can compute in exponential time a $L_\mathbf{S}$ concept, denoted $\mathsf{lub}^\sigma_I(X)$, that is the smallest concept whose extension contains all the elements in X definable in the language. Such concept is polynomial-time computable for bounded schema arity. In particular, the following hold:*

1. *$X \subseteq ext(\mathsf{lub}^\sigma_I(X), I)$,*

2. *there is no concept C' in $L_{\mathbf{S}}$ such that $C' \sqsubset_I \mathsf{lub}_I^{\sigma}(X)$ and $X \subseteq ext(C', I)$.*

By INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS we will refer to the algorithm obtained from INCREMENTAL SEARCH ALGORITHM by replacing $\mathsf{lub}_I(X)$ with $\mathsf{lub}_I^{\sigma}(X)$ in line 3 and line 7.

The following Theorem shows that INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS is correct, i.e., that it outputs an explanation for $\overline{a} \notin Ans$ w.r.t. the $\mathbf{S}$-ontology $\mathcal{O}_I$, and that it runs in exponential time (in polynomial time for bounded schema arity).

Theorem 5.4 (Correctness and running time of INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS)**.** *Let the why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ be an input to* INCREMENTAL SEARCH ALGORITHM WITH SELECTIONS *and E the corresponding output. The following hold:*

1. *E is a most-general explanation for $\overline{a} \notin Ans$ w.r.t. $\mathcal{O}_I = (\mathcal{C}, \sqsubseteq_I, ext)$, where $\mathcal{C}$ is $L_{\mathbf{S}}$;*
2. INCREMENTAL SEARCH ALGORITHM *runs in* EXPTIME *in the size of the input (in* PTIME *for bounded schema arity).*

We close this section with the study of the following problem.

Definition 5.7. *The* CHECK-MGE W.R.T. $\mathcal{O}_I$ *problem is the following decision problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and a tuple of concepts $E = (C_1, \ldots, C_n)$, is E a most-general explanation w.r.t. $\mathcal{O}_I$ for $\overline{a} \notin Ans$?*

Our next proposition states the running time of our algorithm for the CHECK-MGE W.R.T. $\mathcal{O}_I$ for various fragments of our concept language. The algorithm operates very similarly to lines 4-11 of INCREMENTAL SEARCH ALGORITHM. Given a tuple of concepts, we check whether that tuple of concepts can be extended to a more general tuple of concepts through ideas similar to lines 4-11 of INCREMENTAL SEARCH ALGORITHM. If the answer is "no", then we return "yes". Otherwise, we return "no".

Proposition 5.2. *There is an algorithm that solves* CHECK-MGE W.R.T. $\mathcal{O}_I$ *in:*

- PTIME *for selection-free $L_{\mathbf{S}}$, or for $L_{\mathbf{S}}$ with bounded schema arity;*
- EXPTIME *for $L_{\mathbf{S}}$ in the general case.*

5.3 Ontologies from Schema

We now study the case of solving the why-not problem w.r.t. to an $\mathbf{S}$-ontology $O_{\mathbf{S}}$ that is derived from a schema. As in the previous case, the presence of nominals in the concept language guarantees that the trivial explanation always exists. Therefore we do not consider the decision problem EXISTENCE-OF-EXPLANATION W.R.T. $\mathcal{O}_{\mathbf{S}}$.

Definition 5.8 (COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$)**.** *The* COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ *is the following computational problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$, find a most-general explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$ for $\overline{a} \notin Ans$, where $\mathcal{O}_{\mathbf{S}}$ is the $\mathbf{S}$-ontology that is derived from $\mathbf{S}$, as defined in Section 4.2.*

The complexity of COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ depends on the complexity of subsumption checking for $L_{\mathbf{S}}$. As seen in Table 1, subsumption checking with respect to arbitrary integrity constraints is undecidable. Therefore, for the general case in which no restriction is imposed on the integrity constraints, COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ is unlikely to be decidable. The restrictions on the integrity constraints of $\mathbf{S}$ allow for the definition of several variants of the problem that, under some restrictions, are decidable.

We restrict now to the cases in which we are able to materialize the $\mathbf{S}$-ontology $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$, with $\mathcal{K} = \mathrm{adom}(I) \cup \{a_1, \ldots, a_m\}$. EXHAUSTIVE SEARCH ALGORITHM gives us a method for solving COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$. The following proposition gives us a double exponential upper bound for COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ in the general case, and a polynomial case under specific assumptions (cf. Table 1).

Proposition 5.3. *There is an algorithm that solves* COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$

- *in 2*EXPTIME *for $L_{\mathbf{S}}$, provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* EXPTIME *for selection-free $L_{\mathbf{S}}$, and projection-free $L_{\mathbf{S}}$, provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* PTIME *for $L_{\mathbf{S}}^{\min}$, if the arity of q is fixed and provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* PTIME*.*

We end with the definition of CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$.

Definition 5.9. *The* CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ *problem is the following decision problem: given a why-not instance $(\mathbf{S}, I, q, Ans, \overline{a})$ and a tuple of concepts $E = (C_1, \ldots, C_n)$, is E a most-general explanation w.r.t. $\mathcal{O}_{\mathbf{S}}$ for $\overline{a} \notin Ans$?*

As for COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$, the undecidability of concept subsumption in the general case suggests that it is unlikely for CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ to be decidable without imposing any restriction on Π and Σ. However, also this problem allows for the characterization of several decidable variants.

In particular, since CHECK-MGE is solvable in PTIME (see Theorem 5.1), by materializing $\mathcal{O}_{\mathbf{S}}[\mathcal{K}]$ we can derive some upper bounds for CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ too.

Proposition 5.4. *There is an algorithm that solves* CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$

- *in 2*EXPTIME *for $L_{\mathbf{S}}$ concepts, provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* EXPTIME *for selection-free $L_{\mathbf{S}}$, and projection-free $L_{\mathbf{S}}$, provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* EXPTIME*,*
- *in* PTIME *for $L_{\mathbf{S}}^{\min}$, provided that the input schema $\mathbf{S}$ is from a class for which concept subsumption can be checked in* PTIME*.*

The proof is analogous to the one for Proposition 5.3.

We expect that the upper bounds for COMPUTE-ONE-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ and CHECK-MGE W.R.T. $\mathcal{O}_{\mathbf{S}}$ can be improved. Pinpointing the complexity of these problems is left for future work.

6. VARIATIONS OF THE FRAMEWORK

We consider several refinements and variations to our framework involving finding short explanations, and providing alternative definitions of *explanations* and of what it means to be *most general*.

Producing a Short Explanation. A most-general explanation that is *short* may be more helpful to the user. To simplify our discussion, we restrict our attention to ontologies that are derived from an instance and show that the problem of finding a most-general explanation of minimal length is NP-hard in general, where the *length* of an explanation $E = (C_1, \ldots, C_k)$ is measured by the total number of symbols needed to write out $C_1, \ldots, C_k$.

Proposition 6.1. *Given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$*, the problem of finding a most-general explanation to* $\bar{a} \notin Ans$ *of minimal length is* NP*-hard.*

Given that computing a shortest most-general explanation is intractable in general, we may consider the task of shortening a given most-general explanation. The INCREMENTAL SEARCH ALGORITHM produces concepts that may contain superfluous conjuncts. It is thus natural to ask whether the algorithm can be modified to produce a most-general explanation of a shorter length. This question can be formalized in at least two ways.

Let I be an instance of a schema $\mathbf{S}$, and let $C = \sqcap\{C_1, \ldots, C_n\}$ be any $L_\mathbf{S}$ concept expression. We may assume that each C_i is intersection-free. We say that C is *irredundant* if there is a no strict subset $X \subsetneq \{C_1, \ldots, C_n\}$ such that $C \equiv_{\mathcal{O}_I} \sqcap X$. We say that an explanation (with respect to $\mathcal{O}_I$) is irredundant if it consists of irredundant concept expressions. We say that explanations E_1 and E_2 are *equivalent* w.r.t. an ontology $\mathcal{O}$, denoted as $E_1 \equiv_\mathcal{O} E_2$, if $E_1 \leq_\mathcal{O} E_2$ and $E_2 \leq_\mathcal{O} E_1$.

Proposition 6.2. *There is a polynomial-time algorithm that takes as input an instance* I *of a schema* $\mathbf{S}$*, as well as an* $L_\mathbf{S}$ *concept expression* C*, and produces an irredundant concept expression* C' *such that* $C \equiv_{\mathcal{O}_I} C'$*.*

Hence, by combining Proposition 6.2 with INCREMENTAL SEARCH ALGORITHM, we can compute an irredundant most-general explanation w.r.t. $\mathcal{O}_I$ in polynomial time.

We say that an explanation $E = (C_1, \ldots, C_k)$ is *minimized* w.r.t. $\mathcal{O}_I$ if there does not exist an explanation $E' = (C_1, \ldots, C_k)$ such that $E \equiv_{\mathcal{O}_I} E'$ and E' is shorter than E. Every minimized explanation is irredundant, but the converse may not be true. For instance, let O be an ontology with three atomic concepts C_1, C_2, C_3 such that $C_1 \sqsubseteq_O C_2 \sqcap C_3$ and $C_2 \sqcap C_3 \sqsubseteq_O C_1$. Then the concept $C_2 \sqcap C_3$ is irredundant with respect to O. However, C_1 is an equivalent concept of strictly shorter length.

Proposition 6.3. *Given a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an explanation* E *to why* $\bar{a} \notin Ans$*, the problem of finding a minimized explanation equivalent to* E *is* NP*-hard.*

Cardinality based preference. We have currently defined a *most-general explanation* to be an explanation E such that there is no explanation E' with $E' >_\mathcal{O} E$. A natural alternative is to define "most general" in terms of the cardinality of the extensions of the concepts in an explanation. Formally, let $\mathcal{O} = (\mathcal{C}, \sqsubseteq, ext)$ be an $\mathbf{S}$-ontology, and I an instance. We define the *degree of generality* of an explanation $E = (C_1, \ldots, C_m)$ with respect to $\mathcal{O}$ and I to be the (possibly infinite) sum $|ext(C_1, I)| + \cdots + |ext(C_m, I)|$. For two explanations, E_1, E_2, we write $E_1 >^{card}_{\mathcal{O},I} E_2$, if E_1 has a strictly higher degree of generality than E_2 with respect to $\mathcal{O}$ and I. We say that an explanation E is $>^{card}$-maximal (with respect to $\mathcal{O}$ and I) if there is no explanation E' such that $E' >^{card}_{\mathcal{O},I} E$.

Proposition 6.4. *Assuming P≠NP, there is no* PTIME *algorithm that takes as input a why-not instance* $(\mathbf{S}, I, q, Ans, \overline{a})$ *and an* $\mathbf{S}$*-ontology* $\mathcal{O}$*, and produces a* $>^{card}$*-maximal explanation for* $\overline{a} \notin Ans$*. This holds even for unary queries.*

In particular, this shows (assuming P≠NP) that computing $>^{card}$-maximal explanations is harder than computing most-general explanations. The proof of Proposition 6.4 goes by reduction from a suitable variant of SET COVER. Our reduction is in fact an L-reduction, which implies that there is no PTIME constant-factor approximation algorithm for the problem of finding a $>^{card}$-maximal explanation.

Strong explanations. We now examine an alternative notion of an explanation that is essentially independent to the instance of a why-not question. Recall that the second condition of our current definition of an explanation $E = (C_1, \ldots, C_m)$ requires that $ext(C_1, I) \times \cdots \times ext(C_1, I)$ does not intersect with Ans, where I is the given instance. We could replace this condition by a stronger condition, namely that $ext(C_1, I') \times \cdots \times ext(C_1, I')$ does not intersect with $q(I')$, for *any* instance I' of the given schema that is consistent with the ontology $\mathcal{O}$. If this holds, we say that E is a *strong explanation*.

A strong explanation is also an explanation but not necessarily the other way round. When a strong explanation E for $\overline{a} \notin Ans$ exists, then, intuitively, the reason why $\overline{a}$ does not belong to Ans, is essentially independent from the specific instance I, and has to do with the ontology $\mathcal{O}$ and the query q. In the case where the ontology $\mathcal{O}$ is derived from a schema $\mathbf{S}$, a strong explanation may help one discover possible errors in the integrity constraints of $\mathbf{S}$, or in the query q. We leave the study of strong why-not explanations for future work.

7. CONCLUSION

We have presented a new framework for why-not explanations, which leverages concepts from an ontology to provide high-level and meaningful reasons for why a tuple is missing from the result of a query. Our focus in this paper was on developing a principled framework, and on identifying the key algorithmic problems. The exact complexity of some problems raised in this paper remains open. In addition, there are several directions for future work.

Recall that, in general, there may be multiple most-general explanations for $\overline{a} \notin q(I)$. While we have presented a polynomial time algorithm for computing a most-general explanation to a why-not question w.r.t. $\mathcal{O}_I$ for the case of selection-free $L_\mathbf{S}$, the most-general explanation that is returned by the algorithm may not always be the most helpful explanation. In future work, we plan to investigate whether there is a polynomial delay algorithm for enumerating all most-general explanations for such ontologies.

Although we only looked at *why-not explanations*, it will be natural to consider *why explanations* in the context of an ontology, and in particular, understand whether the notion of most-general explanations, suitably adapted, applies in this setting. In addition, Roy and Suciu [27] recently initiated the study of what one could call "why so high" and "why so low" explanations for numerical queries (such as aggregate queries). Again, it would be interesting to see if our approach can help in identifying high-level such explanations.

We have focused on providing why-not explanations to missing tuples of queries that are posed against a database schema. However, our framework for answering the why-not question is general and could, in principle, be applied also to queries posed against the ontology in an OBDA setting.

Finally, we plan to explore ways whereby our high-level explanations can be used to complement and enhance existing data-centric and/or query-centric approaches. We illustrate this with an example. Suppose a certain publication X is missing from the answers to query over some publication database. A most-general explanation may be that X was published by Springer (supposing all Springer publications are missing from the answers to the query). This explanation provides insight on potential high-level issues that may exist in the database and/or query. For example, it may be that all Springer publications are missing from the database (perhaps due to errors in the integration/curation process) or the query has inadvertently omitted the retrieval of all Springer publications. This is in contrast with existing data-centric (resp. query-centric) approaches, which only suggest fixes to the database instance (resp. query) so that the specific publication X appears in the query result.

Acknowledgements We thank Vince Bárány, Bertram Ludäscher and Dan Olteanu for motivating discussion during early stages of the research. Ten Cate is partially supported by NSF grant IIS-1217869. Civili is partially supported by the EU under FP7 project Optique (grant n. FP7-318338). Sherkhonov is supported by the Netherlands Organization for Scientific Research (NWO) under project number 612.001.012 (DEX). Tan is partially supported by NSF grant IIS-1450560.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley, 1995.

[2] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the logicblox system. In *SIGMOD '15*, 2015.

[3] T. Arora, R. Ramakrishnan, W. G. Roth, P. Seshadri, and D. Srivastava. Explaining program execution in deductive systems. In *DOOD*, pages 101–119, 1993.

[4] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69, 2009.

[5] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight linked data publication from relational databases. In *WWW*, pages 621–630, 2009.

[6] A. Baid, W. Wu, C. Sun, A. Doan, and J. F. Naughton. On debugging non-answers in keyword search systems. In *EDBT*, 2015.

[7] M. Benedikt and G. Gottlob. The impact of virtual views on containment. *PVLDB*, 3(1):297–308, 2010.

[8] N. Bidoit, M. Herschel, and K. Tzompanaki. Query-based why-not provenance with nedexplain. In *EDBT*, pages 145–156, 2014.

[9] M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. In *PODS*, pages 213–224, 2013.

[10] C. Bizer and A. Seaborne. D2rq - treating non-rdf databases as virtual rdf graphs. In *ISWC2004 (posters)*, 2004.

[11] A. Borgida, D. Calvanese, and M. Rodriguez-Muro. Explanation in the DL-Lite family of description logics. In *On the Move to Meaningful Internet Systems*, pages 1440–1457, 2008.

[12] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. of Automated reasoning*, 39(3):385–429, 2007.

[13] D. Calvanese, M. Ortiz, M. Simkus, and G. Stefanoni. Reasoning about explanations for negative query answers in DL-Lite. *J. Artif. Intell. Res.*, 48:635–669, 2013.

[14] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[15] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[16] B. Chin, D. von Dincklage, V. Ercegovak, P. Hawkins, M. S. Miller, F. Och, C. Olston, and F. Pereira. Yedalog: Exploring knowledge at scale. In *SNAPL*, 2015. To appear.

[17] F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *EDBT*, pages 561–572, 2013.

[18] T. J. Green. Logiql: a declarative language for enterprise applications. In *PODS '15*, 2015.

[19] T. J. Green, M. Aref, and G. Karvounarakis. Logicblox, platform and language: A tutorial. In *Proceedings of the Second International Conference on Datalog in Academia and Industry*, pages 1–8, 2012.

[20] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[21] T. Halpin and S. Rugaber. *LogiQL: A Query Language for Smart Databases*. CRC Press, 2014.

[22] M. Herschel, M. A. Hernández, and W. C. Tan. Artemis: A system for analyzing missing answers. *PVLDB*, 2(2):1550–1553, 2009.

[23] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[24] L. Lubyte and S. Tessaris. Automatic extraction of ontologies wrapping relational data sources. In *DEXA*, pages 128–142, 2009.

[25] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[26] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics X*, pages 133–173, 2008.

[27] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1579–1590, New York, NY, USA, 2014. ACM.

[28] O. Shmueli and S. Tsur. Logical diagnosis of ldl programs. In *Int'l Conf. on Logic Programming*, 1990.

[29] Q. T. Tran and C. Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26, 2010.